Describing System Functions that Depend on Intermittent and Sequential Behavior

Bell, Jonathan; Snooke, Neal

Publication date:
2004

Citation for published version (APA):
Bell, J., & Snooke, N. (2004). *Describing System Functions that Depend on Intermittent and Sequential Behavior*. 51-57. http://hdl.handle.net/2160/64

Describing System Functions that Depend on Intermittent and Sequential Behavior

Jonathan Bell (jpb@aber.ac.uk) and **Neal Snooke (nns@aber.ac.uk)**
Department of Computer Science, University of Wales Aberystwyth, Penglais,
Aberystwyth, Ceredigion SY23 3DB, United Kingdom

Abstract

Functional modeling languages have been used to describe processes that react to discrete external events and remain in a constant state until another such event stimulates a change in system state, and are deficient in the area of describing several processes occurring in a specific temporal relationship. The lack of such expressiveness means that it is difficult to apply functional modeling effectively to complex systems where such temporal interactions are key to the correct functioning of the system. This paper presents operators to extend the expressiveness of functional modeling for systems that depend on intermittent behavior or on a strict sequence of events. The effectiveness of these operators is shown for characterizing different orderings of behavior, both in cases where the ordering is precisely specified and where some orderings need not be so specified. Their relevance is shown by considering examples from the domain of automotive systems.

Introduction

Functional modeling (Sticklen et al., 1989; Iwasaki et al., 1993) has been in use for a number of years, both for deriving the behavior of a system from knowledge of its structure and the function of the components, and for interpreting the results of a qualitative simulation (in which the system behavior is derived from the system's structure and component behavior and domain rules) in terms of the system's purpose. However, the expressiveness of functional models has been limited to cases where the function is dependent on a continuous state, or at least states that only change in direct response to some change in input. For example, an electrical switch can have either of the functions "conducting" or "isolating", depending on the switch position. This is sufficient to describe systems such as a car headlamp system, where the behavior of the system is that the lamps light in response to them being switched on and they stay lit until they are switched off again. The functionality of many systems depends on responses that are more complex than this, a simple case in point being the two flashes of a car's direction indicators that confirm remote locking of the car. This paper presents operators that allow functions to be combined temporally, allowing such behaviors to be described.

The next section discusses the use and limitations of functional modeling in more detail. It is followed by a description of the operators used for describing sequential functions and a discussion of how these operators are used for describing different orderings of subsidiary functions. Some matters that arise from the use of these operators are then discussed and this is followed by a brief consideration of future work.

The uses of system function

As was suggested in the introduction, much work with functional modeling has been concerned with using knowledge of the function of a system's components to derive the behavior of the system as a whole. This functional knowledge can be used to support various design tasks. A design can be developed by refining a functional model based on the purpose of the system until the functions can be related to individual components (Iwasaki et al., 1993). In this case, the system functions are decomposed in terms of connections between components. Functional modeling has also been used to support diagnosis (Sticklen et al., 1989) and Failure Mode Effects Analysis (Hawkins and Woollons, 1998). In all these cases, system function is expressed in terms of component functions which are related to each other primarily in terms of the connections between components, so capturing the structure of the system. One drawback of this approach, especially for failure analysis such as diagnosis, is that the component's functional models also need functions associated with their failure mode behaviors. Rather than conducting, a wire might isolate (if it has broken), or might conduct to the wrong place (if shorted to power or ground), for example. This naturally complicates the functional models required. The functional decompositions are adequate for systems where the individual component functions are simple (such as a wire conducting) but if a component has a function that depends on behavior of any temporal complexity (such as generating an intermittent output, like the flasher unit for a car's direction indicators), then temporal operators are required, such as those presented in this paper.

Another use of functional knowledge is interpretation of the results of simulation in terms of the purpose of the system as a whole. Either a numerical or qualitative simulation tool can be used to establish the overall behavior of the system being analyzed. Knowledge of the system's functions maps this behavior to the system's purpose, allowing significant changes of behavior to be identified. This is particulary valuable for design analysis tasks such as Failure Mode Effects Analysis (FMEA) where the engineer must generate a report showing the

effects of component failures on the system as a whole (Price, 2000). That report will be couched in terms relating to the intended purpose of the system, so instead of noting that a failure results in no current flowing through a car headlamp, for example, it will note that the headlamp fails to light, and the road will not be lit and the legal implications. This task is a good candidate for automation as it is extremely repetitive and it is best carried out early in the design process, so any changes found necessary can be made easily, and analysis can be done whenever changes are made to the design of the system, so the effects of such changes can be established. While a simulation tool will help with the analysis, the interpretation will still be the task of the engineer. Sneak Circuit Analysis (SCA), in which the system is analyzed to ensure there are no unexpected current flows resulting in unintended system outputs (Price et al., 1996; Savakoor et al., 1993), is another design analysis task where similar arguments apply.

If the interpretation of the simulation is to be automated (so a draft FMEA or SCA report can be produced completely automatically) then some way of mapping the system's behavior to its purpose is needed. One approach to this that has been found to be useful is "functional labeling" (Price, 1998). Functional labels are used to identify the system's outputs and to associate them with the purpose of the system and with the required inputs. These inputs are frequently of information (as to the user's intention for the system) expressed in terms of switch positions (or whatever input interfaces the system provides) rather than of energy. This will be the case where the energy source (a battery for example) is treated as part of the system or in cases where the energy source is external, but can be taken for granted, so analyzing a mains electrical appliance need not extend to an analysis of the public electricity supply! An alternative view of inputs and outputs in this context is as pre- and post-conditions for correct achievement of some system function. In many cases the result of a function's achievement is a goal state rather than an output, as is the case of a car's remote locking system entering the locked state. It is worth noting that for failure analyses (such as FMEA) there might be no need for explicit representation of the input associated with a system function. This is because the failure behavior is compared with the behavior of the system free of failures and this behavior can be used to find the inputs associated with the system functions. This does, of course, rely on the assumption that the behavior free of failures matches the intended behavior.

This approach has been found to work well for design analysis of many electrical systems and is in use in a commercial design analysis tool that allows the automatic generation of FMEA and SCA reports of electrical circuits in the automotive sector. It has the advantage that the functional models are simple, as only those components whose inputs or outputs are also inputs or outputs to the system as a whole need explicit representation in the functional model. The function of other components is (implicitly) derived from their behavior. Also, the functional model has no need of failure mode functions as the component's failure modes are associated with its reusable behavioral model. The functional models are also reusable for systems having a similar purpose, so the functional model for a car lighting system, for example, can be kept and reused for future systems. All that need be changed is the mapping between the component level functions and the state of the actual components.

While this use of functional knowledge does differ from that of other workers in the field, there is common ground. The modeling of function in terms of the inputs to and outputs from a system is not inconsistent with the definition of function as "[a device's] effect on its environment" in (Chandrasekaran and Josephson, 1996). Also the mapping between purpose and behavior and input and output in functional labeling means that the approach specifies the function as the "expected behavior" consistently with the notion of function in (Iwasaki et al., 1993).

Given that a function is considered as a relation between input and output, there are, of course, two ways in which it can fail. The input can fail to result in the expected output (the user throws a switch to "on" with no effect) or the output can occur without the input (a switch is short circuited, leading to the output continuing after the switch is opened).

As the output of a system is a combination of outputs of its components, it is necessary to decompose the system function until the functional model can be mapped to such outputs. This is done as described in (Snooke and Price, 1998) and a simple such hierarchy is illustrated in Figure 1. This hierarchy uses the conventional

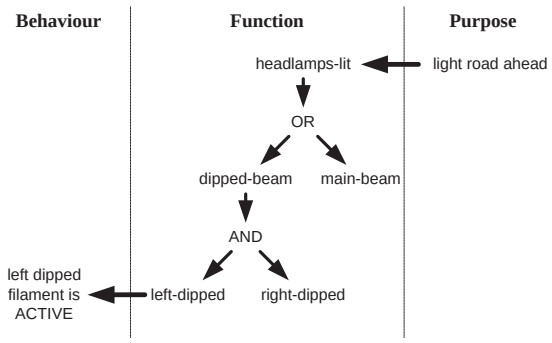


Figure 1: Example hierarchical function for car headlamps

logical relations AND, OR, XOR and NOT to decompose a system function from a top level mapping to purpose down to a bottom level mapping to component output. The use of a functional label to describe an output of a system in terms of its purpose is an abstraction of the system's (intended) behaviour and one of the features of this abstraction is a description in terms of achievement (or otherwise) of the system's purpose. Since any given system state can either be fulfilling some specified purpose or not doing so, using binary relations is an appropriate abstraction for functional labeling. There-

fore one of the features of the mapping from behaviour to purpose might be the reduction of a non binary output to a binary state (achievement or non achievement of the purpose). This might be accomplished by associating the achievement of a purpose with an acceptable range of output values in interpretation of a numerical model of the system concerned. The conventional logical relations are sufficient to describe system functions that result in a constant output that lasts until some change to system input causes the output to change. However, there are many examples of systems whose purpose depends on behavior that results in outputs that are intermittent and sequential. These relations are insufficient for such systems, like the locking confirmation example in the introduction. The additional operators presented below are used to allow the functions of systems that depend on such intermittent and sequential outputs (or behaviors) to be described.

Describing intermittent and sequential outputs

The existing hierarchical functional labels, using the conventional logical relations, are adequate for describing behaviors that are continue until further input triggers a change in behavior. The required simulation can be started by the change of input state (modeling the throwing of a switch, for example) and can end once the system settles into a steady state, when the outputs are checked against the functional model to establish that the expected function is achieved. This means that there is one implicit time step associated with the simulation, as illustrated in Figure 2. This leads to the problem that

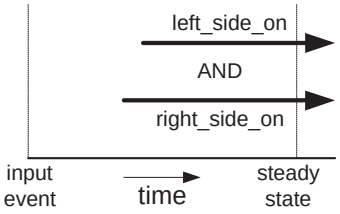


Figure 2: The implicit time step in modeling continuous output

if achievement of a system function depends on some intermittent output that ends before the system reaches a steady state, the output will be missed if the functional model is only compared with the simulation once this has been reached, as illustrated in Figure 3. What is needed is a set of relations between outputs that shows the necessity of checking the simulation against the functional model at intermediate stages of the simulation, allowing the ordering of intermittent and sequential outputs associated with a system function to be described. This amounts to using a relation that shows an output state following some other output state. There are two possible cases here, where the succeeding state should immediately follow the preceding state and where intermediate states can be tolerated. The most important

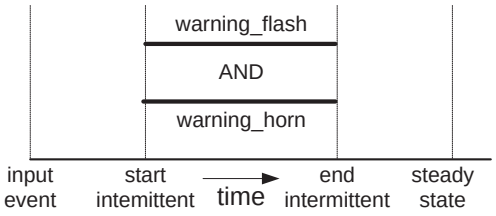


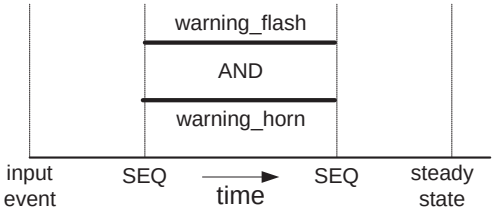
Figure 3: Finding function at the end of simulation might miss significant output

use of the first case is to specify that where the succeeding state includes two or more outputs, they should start simultaneously. To fit these two cases, two relations are used

SEQ The succeeding state is to follow the preceding state with no intermediate states. This is similar to the **O** (in the next time step) operator in some temporal logics.

L-SEQ The succeeding state follows some time after the preceding state. This is similar to the **F** (some time in the future) operator in temporal logic.

The use of a relation associated with the next time slot is necessary to allow simultaneous changes of output to be specified (such as the flashing of a car's direction indicators being synchronized) but it will be appreciated that its use entails a discrete model of time, which is more appropriate in some application domains than others. This model of time is, of course, appropriate for the functional labeling of systems whose simulation is modeled in terms of discrete events, allowing these operators to be used for interpretation of the simulation of systems modeled using state charts. Problems and approaches to this discrete model of time are discussed later. The use of the SEQ relation is illustrated in Figure 4. It will be



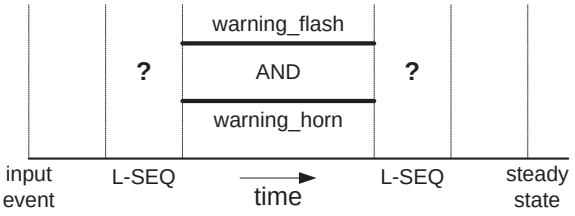
(NOT warning_flash AND NOT warning_horn) SEQ
(warning_flash AND warning_horn) SEQ
(NOT warning_flash AND NOT warning_horn)

Figure 4: Using SEQ to specify two intermittent outputs occurring simultaneously

seen that the use of this relation both specifies the ordering of the state transitions for the outputs associated with a function and also indicates the need to check the outputs at intermediate stages of the simulation, before

the system settles into a steady state. In this case, if (say) the light comes on before the horn sounds, the relation will be false as the succeeding state will not be true in the next time step.

In cases where the outputs need not start simultaneously, the L-SEQ relation can be used, as illustrated in Figure 5. In this case the intermediate states are unde-



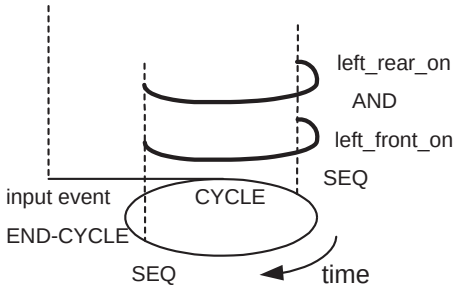
(NOT warning_flash AND NOT warning_horn) L-SEQ
(warning_flash AND warning_horn) L-SEQ
(NOT warning_flash AND NOT warning_horn)

Figure 5: L-SEQ specifies outputs that need not start simultaneously

fined, so either output can start first, or if the output components have several output states (such as a motor running slow or fast) then such outputs might also occur before the specified output. If it is felt necessary to specify the allowable intermediate states, this can be done using the logical relations, so where two outputs are required but need not start together, OR can be used to show that either one can start before the other, but that any other output states are to be excluded.

It is worth noting that as is the case in temporal logic, these operators are unary, and resolve to true if the succeeding relation resolves to true; they are independent of the preceding relation. If that relation was not satisfied then the sequence of outputs would already have failed, of course. This means that the state of the system at the start of the simulation need not be specified. This initial state might not be known when creating the functional model, as it will typically depend on some other system input.

These relations are adequate for describing sequences that terminate with no further system input, so the system will still reach a steady state. However there are cases where a sequence of outputs will continue until some future input causes the output sequence to stop. The flashing of a car's direction indicators until cancelled is a simple example. To indicate such cases, the operators CYCLE and END-CYCLE are used. These enclose a sequence, the END-CYCLE operator simply meaning that the sequence should return to the point indicated by CYCLE. This is illustrated in Figure 6. The outputs that form the sequence to be repeated in the cycle can, of course, be separated either by SEQ or L-SEQ as required. Where this operator is used then the system will not reach a steady state in the simulation step associated with the current input, so the simulator must recognize such cases and end the simulation once such



NOT left_rear_on AND NOT left_front_on
CYCLE SEQ (left_rear_on AND left_front_on)
SEQ (NOT left_rear_on AND NOT left_front_on)
END-CYCLE

Figure 6: Using CYCLE to show a sequence that does not terminate

a cycle has been unambiguously identified. Naturally it needs to recognize the difference between, say, a warning lamp flashing a fixed number of times or it flashing continuously for this recognition to be unambiguous.

Using the sequential operators

In this section we illustrate the use of the sequential operators, both in terms of a simple case study and also for describing different possible temporal relations between required outputs.

A case study

To illustrate how the additional operators can be used to describe the function of systems that otherwise would present difficulties, consider a warning system intended to show that either the driver or front seat passenger of a car has not fastened their seat belt. A typical example of such a system will warn the driver by lighting a telltale lamp on the dashboard and by sounding a chimer several times before stopping the chiming. It will be appreciated that if the simulation is only interpreted in terms of the system's function once a steady state has been reached, this chiming will be lost as the chimer will be silent. Therefore intermediate checks of the system state vis a vis the functional description are needed. Using the proposed operators, the output (post-condition) of the "warning given" function can be described, as

```
‘‘lamp lit’’ AND
(‘‘chimer on’’ SEQ ‘‘chimer off’’
SEQ ‘‘chimer on’’ SEQ ‘‘chimer off’’)
```

assuming two sounds of the chimer are expected.

The use of SEQ shows the intermediate stages of the simulation that must be checked against the functional description, allowing failure of the chimes to be detected as failure to correctly achieve the function. The alternative approach would be to add a specific state to the behavioural model, but this assumes that the behavioural model is presented in some form that allows this (such

as a state chart) and also adds unnecessary complication to that model. A "chiming complete" state could be added, but not only is this state behaviorally identical to the chimer "off state", it is also necessary to add transitions between these two states so that any future achievement of the "warning given" function is correctly detected. Naturally, if the chimer were to fail when the system was already in the "chiming complete" state, this will be missed. The use of the additional operators in the functional description allows greater separation between the different models used in analysis.

The sequential operators and temporal relations

These temporal relations can be combined with the existing hierarchical functional relations to allow various orderings of required outputs to be described. There is a set of thirteen possible orderings of two intermittent states in (Allen, 1984), though twelve of these form six identical pairs. The terms used for the inverse relationships in these pairs are from (Gerevini and Schubert, 1995). These relationships derive from Allen's interval based ontology of time. Any of these can be represented using the relations proposed, as illustrated in Figure 7. It should be noted that the use of NOT is only safe where the state of an output is binary (such as on or off). Some of the labels used in (Allen, 1984) imply some notion of causality which need not be the case. The starting of output y after x, say, need not imply that the starting of y is what causes x to stop. The descriptions using the temporal operators are not intended to express any notion of causality, they are simply concerned with ordering of the output states. The outputs all share the same triggering input. Where functional labeling is used for interpretation of simulation of some system, there is no need for any expression of causality in the functional language, knowledge of causality (if required) being derived from the simulation.

It is possible that the ordering of all the transitions concerned with the starting or stopping of intermittent outputs need not be specified, as it is not felt to be important. If a warning lamp blinks on and off and a chimer sounds, it might be unimportant which happens first. The L-SEQ relation can be used in these cases, as illustrated in Figure 8. L-SEQ is used in all these cases except the first, to show that the expected change to output can take place over more than one transition. The first case differs. Here two intermittent outputs are to occur between the triggering input and the system reaching a steady state but their ordering relative to each other is unspecified. As the transitions indicated by the SEQ relation are local to that sequence, this case can be described by combining two such sequences using the AND relation and there is no ordering implicit between the transitions in the two sequences. This use of AND allows branching of time to be represented, as in such temporal logics as CTL, where it is not the case that some event must always be either before, after or simultaneous with some other event.

It will be seen that in cases where an output is in one of two possible states and there is only one change to the output associated with a sequential relation, then SEQ and L-SEQ are equivalent, but this is not the case where there is some other possible output state that might occur between the two specified states.

While for the sake of simplicity the sequences in the examples have been kept short, it will be appreciated that the required sequences can be of arbitrary length. The example of a system function given in the introduction, the direction indicators all flashing twice simultaneously to confirm remote locking can readily be described using repetitions of the SEQ operator while the outputs of each direction indicator are related using AND.

Discussion

As noted earlier, the use of SEQ (true if succeeding state is true in the next time step) depends on a discrete model of time. This leads to some questions about how time might be represented and also about the presence of intermediate states. As we are concerned with the inputs to and outputs from the system, any intermediate internal states can be ignored for the purpose of determining function, we can restrict our consideration only to those states associated with a change of system output. For example, a lighting system might use a relay whose behavior will include a state in which there is current flowing through the coil (following the throwing of the switch) but in which the relay's switch remains open, so there is a delay in the switching on of the lamps. However, this intermediate change of system state (in which current flows through the relay coil but not the lamps) will not result in any change to the system's output, so can be ignored for assessing the achievement of a system function. It will be appreciated that where two lamps are switched by the same relay, their lighting up can be considered simultaneous (provided there is no difference in their connections resulting in a delay to one or the other). This is appropriate if the behavior of the system can be represented in terms of states and transitions, less so if a continuous model of time is used (as might be the case if the simulation is numerical as opposed to qualitative).

In the case of mechanical systems where there is an inevitable delay between the effort causing an action and the action (because of inertia in the system) there are alternative possibilities. One is that intermediate states are ignored, so a rear window wiper can be modelled as stopped and wiping, ignoring the fact that the system will not reach the working wiping speed instantaneously. If this approach is taken, then operational states can conveniently be modelled using a discrete approximation for modeling time and SEQ can be used (perhaps to describe intermittent wipe). In many cases, this approximation will not result in the loss of any aspect of the behavioral model that relates directly to purpose, so is acceptable. Alternatively, of course, L-SEQ can be used to make explicit the fact that such intermediate states as acceleration or the filling of a tank can be ignored. For example, L-SEQ can be used to describe the situation where opening an inlet valve will eventually result in a

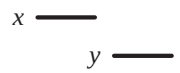
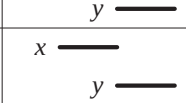
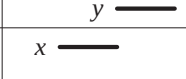
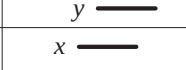
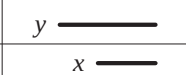
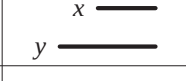
Relation	Meaning	Description
x before *y* *y* after *x*		(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) SEQ (NOT *x* AND NOT *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x meets *y* *y* met-by *x*		(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x overlaps *y* *y* overlapped-by *x*		(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x during *y* *y* contains *x*		(NOT *x* AND NOT *y*) SEQ (NOT *x* AND *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x starts *y* *y* started-by *x*		(NOT *x* AND NOT *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x finishes *y* *y* finished-by *x*		(NOT *x* AND NOT *y*) SEQ (NOT *x* AND *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x equal *y*		(NOT *x* AND NOT *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND NOT *y*)

Figure 7: Using SEQ to describe different temporal relations

reservoir being filled. This means that the use of L-SEQ allows intermediate states that might occur during the transition between different non-binary outputs to be ignored in the functional model. It therefore needs careful use in cases where outputs are not binary as any unexpected outputs will not be noted in the text of the design analysis report. For example, where a system is driven by a motor that might be running slow or fast (such as a wash-wipe system) the use of L-SEQ means that an intermediate period at which the system is running at the unintended speed will be missed. It is worth noting that if function is regarded as a property of the system, the continuously variable nature of outputs of many components can be ignored. In the wash-wipe system, the motor is itself capable of running in either direction at any speed between stopped and its maximum, but the system requires (and constrains) the motor to run in one direction at one of two speeds. As the functional model is concerned with the sweeping of the windscreen wipers, not the output of the motor, this abstraction of its behaviour is adequate. Where a numerical simulation is used, the various output levels (such as motor speeds) can be mapped to the system outputs as a range of acceptable values.

Future work

While the extensions to the functional modeling language proposed herein allow the modeling of intermittent and sequential outputs, there are other aspects of modeling function that might be necessary and which the present work moves towards. One such aspect is mapping these ordering relations to a continuous modeling of time, to model cases where a state transition model is inappropriate. A related aspect of temporal modeling with regard to function is capturing the idea that a system function is achieved, but in an untimely manner (typically later than expected). Some work has been done on this, especially as it relates to the present work, modeling these intermittent behaviors such that the timing of the sequence of transition concerned is specified. The timing of user inputs might also affect the expected behavior of the system. One example that has come to light is the (intended) temporary disabling of a seat belt reminder system by the driver buckling the seat belt and unbuckling it within a specified time.

It is also the case that a system might incorporate a telltale function that informs the user of the state of the system. Such a function, while it contributes to the functionality of the system as a whole, cannot be modelled as part of the hierarchy for the system's 'core' function. For example, the blue dashboard light (showing main beam is selected) does not contribute to the achievement of the headlamp function's purpose of lighting the road ahead, though it shares the input setting for that function. Work is progressing with modeling this class of functional dependency.

Conclusion

The extensions to the functional modeling language proposed herein allow the modeling of intermittent and sequential behaviors, whether they form a sequence that ends with the system in a steady state or the system enters an indefinite cycle. This extension to the language used for defining functional relations appears to be capable of handling such behaviors of arbitrary complexity. This is done in a way that maintains compatibility with the existing relations, allowing existing functional models to continue in use and avoiding any need for additional complexity in the description of simple systems whose functional model can be expressed in terms of the logical operators. It also appears to provide a basis for further extending the language in the ways suggested in

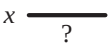
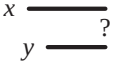
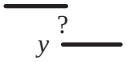
Relation	Meaning	Interpretation	Description
x unordered-with *y*		both *x* and *y* must occur at some stage in the simulation step	(NOT *x* SEQ *x* SEQ NOT *x*) AND (NOT *y* SEQ *y* SEQ NOT *y*)
x must-overlap y		There must be an overlap between *x* and *y* but either can start or end first	(NOT *x* AND NOT *y*) L-SEQ (*x* AND *y*) L-SEQ (NOT *x* AND NOT *y*)
x starts-with *y*		both *x* and *y* must start together but the ordering of the ends is unspecified	(NOT *x* AND NOT *y*) SEQ (*x* AND *y*) L-SEQ (NOT *x* AND NOT *y*)
x ends-with *y*		both *x* and *y* must end together but the ordering of the starts is unspecified	(NOT *x* AND NOT *y*) L-SEQ (*x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x starts-before *y*		*x* must start before *y* but the ordering of the ends is unspecified	(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) SEQ (*x* AND *y*) L-SEQ (NOT *x* AND NOT *y*)
x ends-before *y*		*x* must end before *y* but the ordering of the starts is unspecified	(NOT *x* AND NOT *y*) L-SEQ (*x* AND *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x starts-before *y* and *x* ends-before *y*		*x* must start before *y* starts and end before *y* ends but can end before or after *y* starts	(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) L-SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)

Figure 8: Using L-SEQ to describe sequences where ordering is unimportant

the section on future work.

References

Allen, J. F. (1984). Towards a general theory of action and time. *Artifial Intelligence*, 23(2):123–154.

Chandrasekaran, B. and Josephson, J. R. (1996). Representing function as effect: Assigning functions to objects in context and out. In *Proceedings of American Association for Artificial Intelligence*.

Gerevini, A. and Schubert, L. (1995). Efficient algorithms for qualitative reasoning about time. *Artificial Intelligence*, 74(2):207–248.

Hawkins, P. G. and Woollons, D. J. (1998). Failure modes and effects analysis of complex engineering systems using functional models. *Artificial Intelligence in Engineering*, 12(4):375–397.

Iwasaki, Y., Fikes, R., Vescovi, M., and Chandrasekaran, B. (1993). How things are intended to work: Capturing functional knowledge in device design. In *Proceedings of 13th International Joint Conference on Artificial Intelligence*, pages 1516–1522.

Price, C. J. (1998). Function-directed electrical design analysis. *Artificial Intelligence in Engineering*, 12(4):445–456.

Price, C. J. (2000). AutoSteve: automated electrical design analysis. In *Proceedings ECAI-2000*, pages 721–725.

Price, C. J., Snooke, N., and Landry, J. (1996). Automated sneak identification. *Engineering Applications of Artificial Intelligence*, 9(4):423–427.

Savakoor, D. S., Bowles, J. B., and Bonnell, R. D. (1993). Combining sneak circuit analysis and failure modes and effects analysis. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 199–205.

Snooke, N. A. and Price, C. J. (1998). Hierarchical functional reasoning. *Knowledge-Based Systems*, 11(5–6):301–309.

Sticklen, J., Goel, A., Chandrasekaran, B., and Bond, W. E. (1989). Functional reasoning for design and diagnosis. In *Proceedings Model Based Diagnosis International Workshop (DX-89)*.